The Report Committee for Timothy Vernon Lowery
Certifies that this is the approved version of the following report:

The Autonomous Guidance, Navigation, and Control Laboratory at the University of Texas at Austin

APPROVED BY

SUPERVISING COMMITTEE:

Behçet Açıkmeşe, Supervisor

Maruthi R. Akella

The Autonomous Guidance, Navigation, and Control Laboratory at the University of Texas at Austin

by

Timothy Vernon Lowery, B.S.

Report

Presented to the Faculty of the Graduate School

of The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2015

To all users of the AGNC Lab — past, present, and future

Do cool stuff in here

Acknowledgments

Thank you ...

... **Dr. Açıkmeşe** for the opportunity to learn and grow in your group.

... **Dr. Akella** for serving on my report committee.

... **WRW** for your many helpful, insightful, friendly inhabitants.

... **Undergrads** for all your pieces of the many puzzles.

... **Can** for helping me start a lab from scratch and for hanging out in it.

... **Mark** for only shaking your head a little before setting me straight.

... **Miki** for being a steadfast work buddy and mentor.

... **Mom and Dad** for encouragement and support.

... **Sarah** for the cookies.

I had fun!

Timothy Vernon Lowery

The University of Texas at Austin
December 2015

The Autonomous Guidance, Navigation, and Control Laboratory at the University of Texas at Austin

by

Timothy Vernon Lowery, M.S.E.
The University of Texas at Austin, 2015
Supervisor: Behçet Açıkmeşe

This report details the design, construction, and contents of the Autonomous Guidance, Navigation, and Control Laboratory (AGNC Lab) for Dr. Behçet Açıkmeşe at the University of Texas. It is intended as a resource for those who are new to the lab or to one of its systems. The lab was created to test — on real-world platforms — the control algorithms produced by Dr. Açıkmeşe's research group.

To separate the control problems from other engineering challenges of autonomous vehicles, the lab uses an optical motion capture system which can relay vehicle's their position and orientation. To support hardware development, the lab houses a full compliment of hand tools, electronics equipment, and a 3D extrusion printer.

The primary research vehicle is the quadrotor, selected for its mechanical simplicity, aerial agility, and recent ubiquity. Through the testing of several quadrotors, my group found existing platforms did not fulfill our need for small size and weight, outdoor flight, payload capacity, and computational power.

In response, we designed a custom quadrotor and autopilot. The vehicle flies safely indoors, confidently outdoors, and with a payload of up to half its own mass. The autopilot is based on an ARM microprocessor, leaving ample overhead for our group's algorithms, and can easily add new functionality with breakout boards.

Contents

List of Figures

1 Introduction

Engineers apply knowledge about the physical world to create things that are useful to people. They are the practical companions to the scientists who generated that fundamental knowledge. Working in-between the engineer and the scientist is the engineering researcher, who concerns himself with generating novel ideas that promise to be useful and practical.

Alas, many ideas do not fulfill that promise. An idea may find its flaws anywhere from its inception to its widespread use. One way for an engineering researcher to make a case for the worthiness of his work is to start vetting his ideas as soon as possible: *Can this be accomplished with current technology? Is this economical? Does this stand up to real-world use? What refinements are needed?*

Headed by Dr. Behçet Açıkmeşe, my research group at the University of Texas at Austin had these questions and more. To answer them, we began to build a laboratory for testing.

1.1 Testing Requirements

The group investigates convex optimization, nonlinear and robust control, model predictive control, and distributed control systems. Within these domains, we have a particular focus on giving more autonomy to vehicles, be they on land, in air, at sea, or in space. The lab needs to prove that the resulting algorithms are worthwhile and to help us learn more about them. But how should it be designed?

Isolate the Control Problem In general, autonomy is achieved by many systems working in tandem. (1) Sensors gather information about the vehicle and the environment. (2) Some algorithms use that information to deduce the state of the vehicle and the environment. (3) Only then can control algorithms do the work of deciding the vehicle's actions. However, the first two steps are hardly solved problems. For

instance, GPS can give a vehicle's location with a clear view of the sky but breaks down indoors or amongst buildings. To allow my research group to focus on controls, the lab needed to solve the first two problems in a way that could stand in for other solutions.

Standardize the Test Platform Our algorithms are destined for rockets, airplanes, cars, and boats. Maintaining test platforms for the great variety of vehicles is not feasible for reasons of cost, safety, space, and time. Better to pick a handful of representative vehicles that are easy to work with. Better still to standardize the process of implementing control algorithms on these few vehicles. Then the most effort goes into solving control problems and not constantly modifying things for new tests.

Fit in the Space Dr. Açıkmeşe already had a small room for use as lab space. More space would allow a greater variety of vehicles to operate. The chances of finding a larger space increased as one moved away from the university, however, so did the cost and hassle of conducting tests. Fitting the lab in this space was preferable.

Operate instead of Simulate We could simulate the vehicles and their environments in a computer. This approach has its advantages:

- Uses a minimum of space
- Allows the use of any vehicle or environment that can be simulated
- Solves the isolation problem by allowing perfect knowledge of any state

However it also:

- Can require a huge amount of work to produce a faithful simulation
- Can be difficult to ascertain the fidelity
- Only includes processes you know about

We are interested in proving that our algorithms can work in real-world systems. Our work in convex optimization is specifically motivated by being able to solve optimization problems in real time on computers that run vehicles. We would need to

ensure that even the models of our computers in the simulations are accurate! Simulating for our purpose would entail a painstakingly detailed and validated simulation environment that would still not be guaranteed to reflect reality. Operating a small number of vehicles is an easier path to vetting our ideas.

1.2 Other Labs

Other research groups have similar needs and have built their own labs. These other labs have much in common. They all:

- Operate real vehicles indoors
- Solve the control isolation problem with a motion capture system, having the ability to feed a vehicle its position as measured by external cameras
- Standardize their platform around quadrotors and a few other vehicles, all linked with a standard software interface

Among the most visible labs are the:

- Flying Machine Arena [4, 2] at ETH Zurich
- Multi-Robot Systems Laboratory [5, 1] at the University of Pennsylvania
- Land, Air, and Space Robotics Lab [3] at Texas A&M University

1.3 Our Lab and This Report

After considering our requirements, the Autonomous Guidance, Navigation, and Controls Laboratory (AGNC Lab) began to form. This report is intended as a resource for those who are new to the lab or to one of its systems. It details the design, construction, and contents of the lab, offers lessons learned, and suggests future work. The report is loosely structured to reflect the development of the lab:

Chapter 2 We began by installing a motion capture system to sidestep the sensor and localization problem and to act as a source of truth.

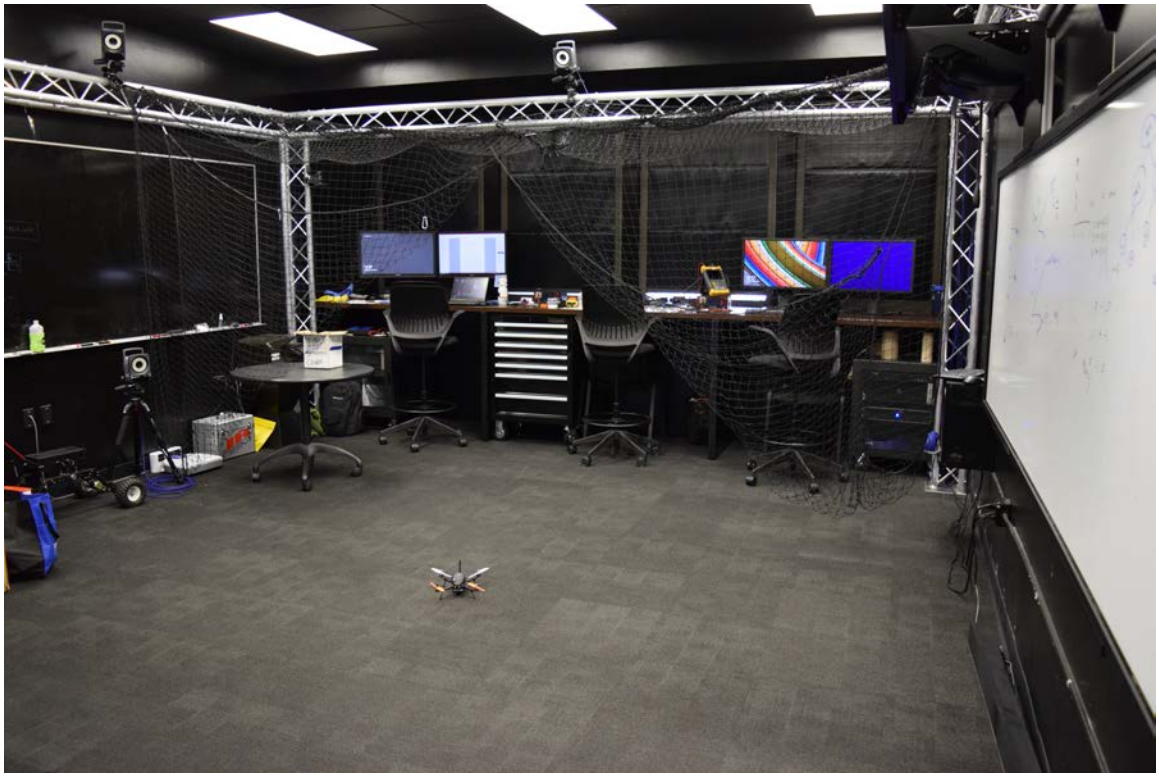

Figure 1.1: The AGNC Lab at time of publication

Chapter 3 We built a computing infrastructure to support the test platform and software development. We assembled prototyping and electronics tools in anticipation of fabricating custom equipment.

Chapter 4 We tested a variety of vehicles and performed experiments with some.

Chapter 5 We realized our ideal test platform did not exist and set about getting as close as we could with our own design.

2 Motion Capture System

We wanted a motion capture system to decouple the problems of localization and attitude determination from the problems of control, letting us focus on that aspect which is the core of our research. The system can also act as a source of truth for localization and attitude determination when they are brought into the loop.

2.1 Product Research

Motion capture systems can operate in many ways, but our requirements filter the options quickly. Electro-magnetic time-of-flight systems are not yet feasible for distance scales we are interested in (millimeters). Acoustic systems are rare and subject to noise, which propeller-driven vehicles generate plenty of. Magnetic inductance systems have a short range. Inertial sensors are subject to measurement drift.

The most popular systems use cameras to optically track the positions of markers and infer the orientation of rigid bodies from multiple markers. We focused on the big three names in optical motion capture systems — OptiTrack, PhaseSpace, and Vicon — because of their proven track record in other labs.

Passive vs Active Among the optical motion capture systems, there are two approaches. Some systems use passive reflective markers that are illuminated by emitters. Other systems use active markers that emit their own light or even modulate their emitter to send information.

Active markers have one huge advantage over passive markers: the information encoded in their flashing can be used to identify them uniquely. Passive systems rely on unique "constellations" of markers to distinguish different objects. All objects with the same arrangement of markers look identical. Objects in active systems can all use the same marker constellations and still be identified uniquely. For whole swarms of vehicles, this is attractive from a logistical point of view.

Unfortunately, active markers have a huge downfall: mass. As sold, the active markers require a large box to drive them that would be difficult to mount on small vehicles and impossible to lift for small air vehicles. Passive markers are small, light, and flexible in their mounting. Mass put the only active system, PhaseSpace, out of the running.

Frame Rate and Resolutions The differentiating factors between the remaining two systems were frame rate and resolution of the cameras. Both act to give greater positional accuracy and smoother tracking. The Vicon cameras outperform the OptiTrack cameras in this area.

2.2 Vicon System

The lab is outfitted with 10 Vicon cameras. Eight of the cameras are mounted to an aluminum truss, while two remain mobile with tripods to handle difficult obstacle geometries. The system can deliver sub-millimeter accuracy at 500 Hz.

Method of Operation The cameras watch for spherical retroreflective markers, illuminated by a ring of near-infrared LEDs around the lenses. The cameras then calculate the centroids of the markers in view and pass the 2D coordinates to a central processing unit, called Giganet.

To make sense of the centroid data, Giganet needs to know where the cameras are. It gets this information through a calibration routine. A wand containing IR LEDs in a known arrangement is moved throughout the capture volume. With enough sightings of the wand, the system can infer where the cameras are. Knowing the positions of the cameras, Giganet can solve the inverse problem to identify the 3D coordinates of the markers and pass them to a PC workstation.

The Tracker software runs on this workstation and acts as an interface to the system. Using it, constellations of markers are defined that represent objects. The constellations must be asymmetric and unique for the system to identify and track objects' positions and orientations.

Cameras The cameras consist of: the body, which houses the sensor and electronics; the lens; and the IR (780 nm) emitter. The cameras are model T40S with a resolution

Figure 2.1: Vicon cameras

of 4 megapixels and a maximum frame rate of 500 Hz. The frame rate can be increased beyond 500 Hz but at the cost of field of view, as the cameras can scan only a fraction of their sensors at higher rates.

We have two sets of lenses — 12.5 mm and 18 mm — to adjust the field of view of the cameras for different size capture volumes. They are paired with appropriate IR emitters.

2.2.1 Accessing Vicon Data

While position and orientation data can be displayed and graphed in Tracker, the real power of a motion capture system is in integrating it with some other system. As of now, there are several ways to access Vicon data, depending on your needs.

Vicon DataStream SDK The Vicon system comes with the DataStream SDK: a set of C++, MATLAB, and LabView libraries that connect to the Tracker software over a TCP network connection. The SDK gives access to the position and orientation of objects, as well as marker positions and other system parameters. Object information is accessed through the name given to an object in Tracker.

Unfortunately, the binaries provided by Vicon are limited to the x86/x64 architectures, so ARM processors and microcontrollers are off limits. Additionally, the Windows binaries were compiled with Visual Studio 2012. Due to name mangling, this means any Windows program using the SDK must be compiled in Visual Studio 2012.

This problem can be addressed by writing a new interface to expose the SDK and compiling it only once in Visual Studio. The wrappers discussed below are two examples. A more flexible solution would be to wrap each function of the SDK in a C-style interface that will not be subject to name mangling.

Python Wrapper The C++ SDK has been wrapped with a Python module for access to data inside of Python programs. The wrapper currently only contains three functions: connect to Tracker, disconnect, and retrieve data. It accesses a fixed set of information for objects named "Quad1" - "Quad4." This was sufficient for the task it was created for but is very limited.

For more functionality, all of the SDK functions could be wrapped. This can be done manually as in the current wrapper. There are also tools to make this easier. An excellent discussion of options, including SWIG, pyrex, and SIP, can be found here.

vicon-server The vicon-server software goes a step further and changes the SDK interface to a set of UDP packets. The packets bypass the architecture limitations and some possibly undesirable behavior: that the SDK sends all information to every client. This consumes network bandwidth and forces every client to process all the information. Vicon-server provides more control over the networking.

The server can be run on any computer networked with the workstation running Tracker; however, it is recommended to run it on the same workstation to minimize latency. The server is configured with a JSON file before startup. The file defines what object names to look for, where their data should be sent, and how often. Position, synthetic compass heading, and an attitude quaternion are currently sent to each entry in the file.

vicon-server should be extended to accept a request packet for certain information on a certain object. For example, vehicles could then send a request for their position at a certain update rate. The server is also an appropriate place to implement a

logger for saving object information. Lastly, all of this should be tied up nicely into a tray icon and GUI that are easier to use than a config file.

2.2.2 Using Vicon

This section only contains the most hard-earned information and ideas about using Vicon.

- For a walk-through of the basics, please see the "How To" video.
- For detailed use information, see the Tracker help files.
- For software and updates, visit Vicon Downloads.

Creating Marker Constellations

- Even though most of the cameras are near the ceiling, markers on the underside of objects help maintain tracking during maneuvers.
- Fitting markers on small objects is difficult. Tracking many small objects has not been tried.
- Someone should investigate whether setting markers to "required" in an object improves tracking at the fringes of the capture volume and whether it helps when similar objects are visible at the same time.

Calibrating

- When masking the cameras before calibration, there is no reason to mask more of the cameras' views than you have to: remove any markers from view and minimize reflections before auto-masking.
- When calibrating, try to pass the wand through the entire capture volume at many angles. This improves the quality of the calibration.
- Calibration will drift over time. If tracking looks bad, try recalibrating.

Miscellaneous Tips

- If Tracker is not connecting to Vicon, check for another instance running under another user.

3 Other Lab Equipment

3.1 Computers and Networking

The lab contains two Dell Precision T7600 workstations with 16 cores and 32 GB of memory apiece. One acts as a server and hosts virtual machines. The other is for interfacing with lab equipment. Another computer runs the projection system. Two Thinkpad docking stations are available for lab laptops. All of these are connected to the UT network through a switch.

Equipment Workstation This Dell is connected to the Vicon Giganet, so it runs the Vicon Tracker software. The workstation is also connected to the router that is the WiFi access point for the vehicles, so it can communicate with them directly. This makes it the appropriate computer on which to run vicon-server and any off-board processing for the vehicles.

Server Workstation The other workstation is a Windows Server 2012 Hyper-V host for virtual machines. It is not intended to be accessed physically except for maintenance. Nothing should be installed or used from the Windows Server installation; this will slow the performance of the guest machines.

agnc-store A Linux RedHat VM that hosts the git repositories for the lab. It could also host a Samba file share or website to disseminate lab information to users.

The VM is connected to the utexas domain, so anyone with a UT EID can be added to an access list and log in with their password. Permissions for repository and file access are configured per user.

agnc-dev Intended for building any software that would take too long to compile on a slower machine. It could also be used as a preexisting Linux build environment

for anyone who does not want to set up their own.

Version Control with git A version control system is a set of tools for tracking changes to files. git is our version control system of choice in the lab. You may have used SVN, an older system. git is different in some important ways:

- git is distributed, i.e. everyone has a copy of the repository, so you can work offline and things happen fast
- git is all about a branch-based workflow

Our git repositories are hosted on agnc-store.

Projection System Along one wall of the lab is a projection system and the computer that runs it. The screen is useful for meetings. It can double as a marker board and has infrared sensors along its border that detect touch. Beyond meetings, the screen was acquired to be used as a vision recognition target, although it has yet to serve the purpose. The intent was to have a dynamic and repeatable source of imagery for testing vision algorithms on-board vehicles.

3.2 Manufacturing and Testing

The lab was outfitted with the expectation that testing would require custom hardware. We have a range of hand tools and measuring devices for working on small parts. There are electronics tools including a soldering station, power supplies, battery chargers, and a digital oscilloscope.

3D Printer The most exciting prototyping tool in the lab is arguably our Stratasys Fortus 250mc 3D extrusion printer. It prints in ABS plastic with layers as fine as 178 micrometers. The build volume is 10 x 10 x 12 inches. The interior of parts can be automatically latticed to reduce weight. The printer is responsible for several parts in our custom quadrotor.

4 Vehicles

As our representative air vehicle, we soon settled on the quadrotor. Quadrotors have become the vehicle of choice for indoor control labs. And for good reason, quadrotors:

- Have exploded commercially, dropping the price of the vehicles and components
- Are mechanically simple, so they are difficult to break and easy to repair
- Are dynamically interesting, being an inherently unstable vehicle
- Operate around a hover, so they can be flown in small, indoor spaces
- Form a decent analog to rockets for our purposes

The next step was to familiarize ourselves with the quadrotors on the market.

4.1 Product Research

There are a few broad classes of quadrotor available. Toy quads are cheap and light, built for entertainment and enthusiasts. Commercial quads are more expensive and rugged, built for working environments and larger payloads. Development quads are targeted at the hobbyist and research market, with more documentation and access to the hardware and electronics.

We purchased a sampling of toy and development quads. The toy quads were used mainly to get a feel for the handling qualities of different sized vehicles as well as their safety indoors. With the development quads, we were hoping to find a usable off-the-shelf platform that we could begin integrating with the motion capture system and our control algorithms.

4.2 AscTec Hummingbird

The Ascending Technologies Hummingbird is used by ETH Zurich and the University of Pennsylvania. It is 76 cm in diameter. The autopilot is broken into two processors: one which runs user code and one which always runs AscTec's code as a backup.

We purchased a Hummingbird with an additional CPU and camera for running vision algorithms. With bumpers surrounding the propellers, the vehicle feels safe indoors; however, its mass keeps it from being aggressively maneuverable in our lab space. The software environment around the Hummingbird is quite friendly. I recommend it as a start for any future machine vision work, but its high unit cost and size keep it from being our main quadrotor.

4.3 Crazyflies

If the AscTec Hummingbird is a large, expensive vehicle for research institutions, the Crazyflie is the opposite. It is an open source hardware and software project from Bitcraze AB intended for anyone to tinker with. The vehicle is 9 cm in diameter, weighs 19 g, and flies for 7 minutes. It is run by a 72 MHz microcontroller. With its size and cost, the Crazyflie is a candidate for aerial swarm research. However, a few hurdles need to be cleared before using it in our lab.

Can Pehlivantürk flew a Crazyflie, with position feedback from Vicon, using an offline trajectory generator [6]. From that process, we learned important lessons about the Crazyflies:

- Attaching the Vicon markers to something that small is difficult.
- Four markers near the lift capacity, giving it little control authority.
- The vehicle could be disturbed by air currents from the open door or air handler.
- The communication link would often disconnect without any obvious cause.
- Communication with multiple vehicles was not an out-of-the-box feature (may have changed in recent software updates).

If these problems could be solved or mitigated, Crazyflies would make an economical swarm.

4.4 Custom Quadrotor

See Chapter 5 for the rationale and development of our custom quadrotor.

4.5 Ground Vehicles

While not receiving the same amount of attention as air vehicles at the inception of the lab, we have considered some ground vehicle options. Ground vehicles can be cheaper and operate longer than air vehicles. They also present their own unique control challenges.

Kilobots The Kilobot is designed for swarm research. To reduce size and cost, it skitters on three prongs by driving two vibration motors. It communicates with its neighbors via a downward-facing infrared LED and photodiode.

We have a starter set of three Kilobots. They have only been demonstrated running their built-in test program. More experimentation is required before knowing how they might be used.

Custom Ground Vehicle We are interested in having a ground vehicle that also uses the MikiPilot board (see Section 5.2). The Kilobots are too small to accommodate it. One option is to build our own vehicle.

UT aerospace undergraduate Felix Zhang has prototyped an omni-directional vehicle using our 3D printer. Four servos drive wheels with built-in casters, allowing independent translation and rotation.

Neato Botvac The Botvac is a robotic vacuum cleaner that has been keeping the lab clean. It is not a promising candidate to become the ground vehicle of choice for the lab, because it large, expensive, and lacks a convenient interface.

However, the Botvac is not uninteresting. As demonstrated by visiting high school student Vishrudh Sriramprasad, it can be commanded with an ASCII serial protocol through a USB port. The vacuum navigates with a bumper and a spinning infrared laser. The data from these sensors is also exposed through the serial protocol. With some hardware modifications, the vacuum could be programmed or controlled remotely.

5 Custom Quadrotor

From our experience with the Crazyflies, we knew that we wanted a larger vehicle. A vehicle that:

- Could operate outdoors
- Had the computing power to run complex algorithms
- Would be easy to mount Vicon markers to
- Simplified the networking problem
- Could lift more — i.e. heavier — payloads

5.1 Vehicle Design

The vehicle design began with testing a variety of off-the-shelf quadrotors of differing sizes. These were flown manually inside the lab and outdoors. From the experience, we settled on a chassis diameter of approximately 25 cm. This size produces a light vehicle with small propellers and low tip velocities. Vehicles of this size felt comfortable in the lab but had the mass and thrust to fight wind disturbances outdoors.

5.1.1 Propulsion

Propellers With a chassis diameter of 25 cm, the maximum propeller diameter is 7 inches. We tested 4, 5, and 6 inch propellers to leave room for payload in the center of the chassis. Plastic and carbon fiber props were tested.

Motors Toy quadrotors at this scale tend to use cheap motors. We wanted the extra efficiency and performance of a brushless motor. For the projected 200 g mass, there were only a handful of motors with appropriate power and reasonable efficiency.

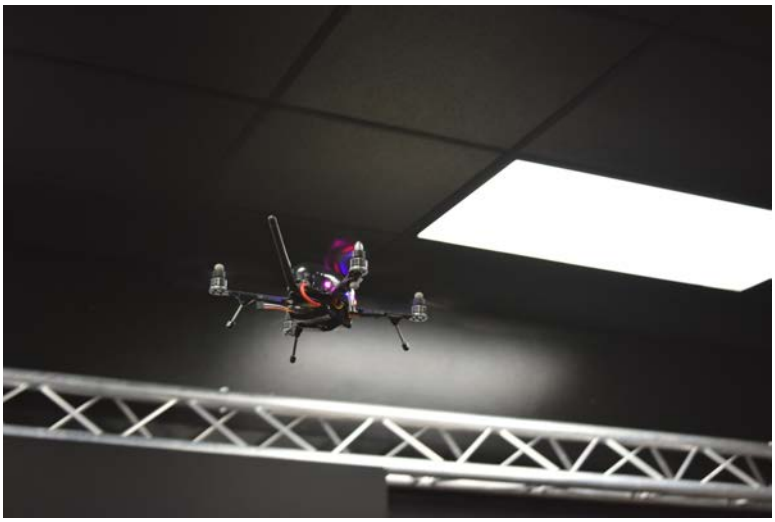

Figure 5.1: The custom quadrotor in flight

ESCs The electronic speed controller turns a command PWM signal into power pulses to drive the brushless motor. From flying test vehicles with various ESCs in our power class, we found ESCs with a higher refresh rate produced more stable flight under disturbances. We chose the ESC with the highest refresh rate that satisfied our voltage requirements. It also happened to be the lightest.

Combinatorial Testing In testing, we were looking for the propeller, motor, and ESC combination that gave the best efficiency across our operating thrusts. Combinations were tested for static thrust on a hinged boom resting on a scale. Thrust and RPM were measured for a given amp draw. One combination with a carbon fiber propeller came out above the rest. The same motor and ESC with a plastic prop was slightly less efficient and provided slightly less max thrust. We chose to design around the plastic prop because the sharp, stiff carbon props are not safe to fly indoors.

Battery Selection Among the most energy dense batteries available, we tested several capacities for hover performance and excess power. For general use, we settled on a 3-cell 2000 mAh lithium polymer battery.

5.1.2 Chassis

Frame The frame is cut from a single pre-impregnated carbon fiber plate. The center is an ellipsoidal platform to provide mounting for rectangular electronics.

Legs The legs consist of two 3D printed parts — a mounting base and a protective nub — joined by a carbon fiber rod. The legs are attached to through-holes in the chassis by a pair of nuts and bolts.

Mounting Electronics or other payloads can be mounted above and below the chassis with shelves and standoffs. Several hole patterns provide options. Circuit boards and shelves can be stacked for modular arrangements. The battery is secured underneath the chassis with a Velcro strap.

Cover The cover slips over the front edge of the chassis and secures with one screw. There is a cutout for the battery connector and space around the perimeter for wiring to exit. The cover serves several functions:

- Protects the payload
- Holds the antenna housing in place
- Streamlines the vehicle
- Houses a strip of RGB LEDs

5.2 MikiPilot

MikiPilot is a hardware and software ecosystem for controlling autonomous vehicles. It is the brain child of Michael Szmuk, a member of our research group. The second generation of MikiPilot is being developed alongside our in-house quadrotor.

MikiPilot is clocked ten times faster than the Crazyflies and comes with all the advantages of running a full Linux operating system. The hardware is small, light, and extensible.

5.2.1 Hardware

The MikiPilot hardware aboard our quadrotor can be broken into three pieces: an off-the-shelf computer on a module, a custom autopilot board, and a breakout board specifically for our quadrotor. The pieces snap together via connectors that link the circuits.

Gumstix Overo The heart of MikiPilot is a Gumstix Overo computer-on-module. The module is based on an ARM Cortex-A8 clocked at 800 MHz with 512 MB RAM. It has onboard WiFi and Bluetooth radios for communication. A 512 MB NAND chip and micro SD card reader are available for storage. By way of comparison, these specifications are similar to an iPhone 4 from 2010. The whole computer weighs 4.5 g and is about the size of a stick of gum, hence the name.

MikiPilot Board The MikiPilot board provides power, sensors, and connectors to the computer which snaps into the top face of the board. The sensors are an inertial measurement unit (IMU) with 3-axis gyros and accelerometers, a magnetometer, and an altimeter. Connectors around the perimeter are available for external sensors, motor control signals, servo control signals, and RC inputs. Connectors on the bottom of the board can be used to exchange power, signals, and communication with external boards.

With an RC receiver plugged in, MikiPilot can interpret and act on the incoming signals or pass them through to some other device, such as a separate stabilizer.

Quad Board The Quad board plugs into the bottom of MikiPilot and specializes it for operating our quadrotor. The Quad board:

- Receives, regulates, and monitors battery power
- Provides convenient connections to motors and servos
- Drives a strip of RGB LEDs
- Holds a coin cell battery to keep the clock alive on the computer

Peripherals Between the PWM inputs/outputs, SPI bus, I^2C bus, UART, and several digital pins, MikiPilot exposes many ways to add functionality. The existing peripherals are:

- An RC receiver and separate stabilizer for use by a safety pilot
- An external GPS and compass unit for autonomous flight outdoors
- A strip of individually controlled RGB LEDs for visual feedback

Some possible future peripherals are:

- Another computer that runs computationally intense code for vision
- A downward facing sonar module for outdoor landings
- A stabilized gimbal for a camera

5.2.2 Software

The MikiPilot software consists of: the flight controller (FSW) which flies and manages the vehicle; the ground control station (GCS) which commands and displays information about vehicles; a data plotter; and a data logger. Being written from scratch, the software is tailored for research use. Modifications can be made to any layer in the system, and new layers can be added on top.

FSW runs on the vehicle in MikiPilot's embedded Linux operating system. Using a full Linux OS, instead of the stripped down operating systems that run on microcontrollers, is a boon to software development. Changes to code can be made directly on board the vehicle and compiled there with no firmware flashing process. The entire ecosystem of Linux drivers and tools is available for adding functionality or troubleshooting.

Programs not running on the vehicle are cross-platform and communicate with vehicles via UDP packets sent over WiFi.

5.2.3 Future Development

Yocto-based Operating System Image The Gumstix Overo is running a Linux operating system from 2011. It is no longer supported by the manufacturer, does not

have access to newer and updated software packages, and exhibits some finicky WiFi behavior.

Gumstix now uses the Yocto Project build system to produce a custom-tailored Linux operating system for the Overos. A Yocto build is defined by a hierarchy of "recipes" that define the components of the operating system and anything installed alongside it. Gumstix makes their recipes public so that users can tweak and modify the software environment on their Gumstix COMs.

Using this system with MikiPilot has tangible advantages:

- Support from the manufacturer
- Automated package generation, allowing MikiPilots to request updates with a standard Linux package manager
- Automated cross-compiler toolchain generation, producing a binary that installs the full environment necessary to compile for MikiPilot off-board, i.e. much faster
- Easy changes to the Linux kernel to streamline or add drivers

The downside to all of this control is a steep learning curve. I found that the Yocto build from Gumstix is not a drop-in replacement for the 2011 version. When I first booted it, WiFi did not work properly, some drivers were not enabled, and GPIOs were not configured the same. I have found fixes for all of those problems by changing the build recipes but more have cropped up to take their place.

The idea remains a tempting way to solve the problems of software updating and cross-compilation.

Node Discovery Currently, IP addresses are input when starting programs and all communication ports are predefined. This static addressing is simple but has drawbacks. Manual entry of IP addresses is tedious and subject to error. With fixed ports, programs that use the same communication channels cannot coexist on the same computer, e.g. simulating multiple vehicles on a computer running GCS. Instead, I propose a protocol to allow programs to discover each other over the local network.

Each program is a node. A node only needs to know the network address and a discovery port number. The network address will only change when the network

changes and, even then, will often stay the same between routers. The discovery port should be a port that is generally free and can always be bound to.

When a node starts, it binds to both the discovery port and some unused port (call it the comm port) and begins to listen. It then starts to periodically send a discovery packet to the broadcast address. The discovery packet contains:

- A unique bit string identifying it as a discovery packet, preventing confusion if other packets come in on the public port
- The port number bound at start up, to allow other nodes to communicate directly (this does not need to be explicitly included in the packet if the discovery packets are sent from the comm socket)
- A node type identifier or list of identifiers, to let other nodes know what services this node provides

When a node receives a discovery packet, it can contact the sending node about services or ignore the packet if it has seen that IP and port before. A timeout could govern node aliveness; for a broadcast interval of 1 second, perhaps nodes that fail to report in after 2 seconds could be removed from the list of active nodes.

For systems involving many vehicles and programs talking to them, a distributed discovery approach could remove significant burden in the connecting and reconnecting process.

Packet Metadata In MikiPilot, information is passed inside programs and over the network via packets. These packets contain variables and methods to operate on them. The only way to access a member variable of a packet is to know that it exists and to know its name at compile time. This presents a problem for other code when packets change their contents or are created. For instance, a code that sets the values in a gains packet must be updated if the number of gains changes. A solution is to provide a more generalized, alternative interface such as a function that accepts a container of values.

Bibliography

[1] *Facilities—GRASP Lab*. Penn Engineering GRASP Laboratory. 2015. URL: https://www.grasp.upenn.edu/facilities (visited on 12/01/2015).

[2] *Flying Machine Arena*. ETH Zurich. 2015. URL: http://flyingmachinearena.org/ (visited on 12/01/2015).

[3] *LASR Laboratory*. LASR Laboratory, Texas A&M University. 2015. URL: http://lasr.tamu.edu/ (visited on 12/01/2015).

[4] Sergei Lupashin et al. "A platform for aerial robotics research and demonstration: The Flying Machine Arena". In: *Mechatronics* 24.1 (2014), pp. 41–54. ISSN: 0957-4158. DOI: http://dx.doi.org/10.1016/j.mechatronics.2013.11.006. URL: http://www.sciencedirect.com/science/article/pii/S0957415813002262.

[5] Nathan Michael et al. "The GRASP Multiple Micro-UAV Testbed". In: *Robotics Automation Magazine, IEEE* 17.3 (Sept. 2010), pp. 56–65. ISSN: 1070-9932. DOI: 10.1109/MRA.2010.937855.

[6] Can Pehlivantürk. "Lossless convexification of quadrotor motion planning with experiments". Thesis. University of Texas at Austin, Aug. 2014. URL: http://hdl.handle.net/2152/26388.